Noise-contrastive estimation of normalising constants and GANs

Contents

1 Fonctions génériques

```
library(ggplot2)
library(reshape)
library(matrixStats)
library(knitr)
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
## L'objet suivant est masqué depuis 'package:matrixStats':
##
##     count
## L'objet suivant est masqué depuis 'package:reshape':
##
##     rename
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

1.1 Algorithme d'Hasting

Utilité : simuler selon $p_m(., \psi)$ pour un paramètre ψ choisi.

Argument	**Type**	**Exemple**	**Indication**
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
n	entier	100	taille de la simulation
psi	vecteur	c(0,1)	paramètres de la fonction h
h	fonction		fonction qui retourne $\overline{p_m}(., \psi)$

```
hasting = function(x, n, psi, h){
  y = c()
  y = append(y, sample(sample(x, 1)))
  for (i in 2:n){
    y_ = rnorm(1, y[i-1], 1)
    u = runif(1)
    if ( u <=
         (h(y_,psi) * dnorm(y_, y[i-1], 1))
         /(h(y[i-1],psi) * dnorm(y[i-1], y_, 1))
    ){
      y = append(y, y_)}
    else {
      y = append(y, y[i-1])
    }
  }
  return (y)
}
```

Note : on peut très certainement écrire sous forme matricielle cette fonction pour une meilleure performance.

1.2 MC MLE

Utilité : retourne une estimation des paramètres selon la méthode décrite dans le papier de Geyer.

```
mc_mle = function(x, n, psi, h){

  m = length(x)

  y = hasting(x, n, psi, h)

  L = function(theta){
    return(sum(log(h(x,theta)/h(x,psi))) - m*log(mean(h(y,theta)/h(y,psi))))
  }

  theta = optim(
    par = rep(1,length(psi)),
    gr = "CG",
    control = list(fnscale=-1),
    fn = L
  )$par

  return(theta)
}
```

1.3 NCE

Utilité : Retourne l'estimation de la constante et des paramètres.

Argument	Type	Exemple	Indication
x	vecteur	rcauchy(100, 0, 1)	notre échantillon de densité inconnue
law_y	fonction	rnorm	fonction qui retourne un échantillon suivant la loi p_n
n	entier	100	taille de l'échantillon de bruit suivant la loi p_n
params_y	vecteur	c(0,1)	arguments de la fonction law_y
log_pm	fonction		fonction qui retourne le logarithme de la densité p_m
log_pn	fonction		fonction qui retourne le logarithme de la densité p_n
size_theta	entier	3	taille de θ, vaut habituellement 2 ou 3
method	string	"CG"	méthode d'optimisation, habituellement "CG" ou "BFGS"

```
nce = function(x, law_y, params_y, log_pm, log_pn, size_theta, n){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  h = function(u, theta){
    return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
  }

  J = function(theta){
    return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
  }

  theta = optim(
    par = rep(1, size_theta),
    gr = "CG",
```

```
    control = list(fnscale=-1),
    fn = J
  )$par

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

1.4 Graphiques

Utilité : afficher l'histogramme pour un échantillon de données x.

```
print_hist = function(x) {
  df = data.frame(x = x)
  hist_x = ggplot(df, aes(x=x)) +
           geom_histogram(aes(y = stat(count)/sum(count)), bins = 20, color="white") +
           theme(aspect.ratio = 1) +
           labs(y = "Fréquence") +
           ggtitle("Distribution de l'échantillon x")
  print(hist_x)
}
```

Utilité : pour NCE, afficher l'évolution des paramètres au fur et à mesure de l'augmentation de n (la dimension de l'échantillon de bruit)

```
NCE_evol_params = function(x, law_y, params_y, log_pm, log_pn, size_theta, ratio, steps, labels) {

  # Creation de l'abscisse
  m = length(x)
  N = seq(0, m*ratio, length.out = steps + 1)

  # Creation de l'ordonnée
  theta = c()
  for (n in N) {
    theta = append(theta, nce(x, law_y, params_y, log_pm, log_pn, size_theta, n))
  }

  # Formatage des données
  theta = t(rbind(matrix(theta, nrow = size_theta),N))
  df = as.data.frame(theta)
  df_melted = melt(df, id.vars = "N")

  # Plot
  plot_df = ggplot(df_melted, aes(x = N, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  geom_point(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par rapport au bruit",
       x = "n (taille du bruit)",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))

  print(plot_df)

  #return(theta)
}
```

Note : il faudrait optimiser le temps de calcul de ces fonctions, peut-être en matriciel au lieu des boucles ou bien avec du calcul en parralèle sur CPU/GPU

2 Applications

2.1 Exemple basique : la loi normale

Soit x l'échantillon de taille m obtenu selon la loi de densité inconnue p_d.

On considère ici que p_d appartient à la famille de fonctions paramétrées par $\theta = (c, \mu, \sigma)$ suivante :

$$p_m(u;\theta) = \frac{1}{Z(\mu,\sigma)} \times exp\Big[-\frac{1}{2}\big(\frac{u-\mu}{\sigma}\big)^2\Big] \quad \text{d'où} \quad ln(p_m(u;\theta)) = c - \frac{1}{2}\big(\frac{u}{\sigma} - \frac{\mu}{\sigma}\big)^2$$

```
pm_barre = function(u, theta){
  return(exp(-0.5 * ((u - theta[1]) / theta[2]) ** 2))
}

log_pm = function(u,theta){
  return(theta[3] - 1/2 * (u/theta[2] - theta[1]/theta[2]) ** 2)
  # theta[1] = mu / theta[2] = sigma / theta[3] = c
}

log_pn_cauchy = function(u){
  return(log(dcauchy(u, mean(x), sd(x))))
}

m = 10000
n = 10000
x = rnorm(m, 2, 4)
size_theta = 3
```

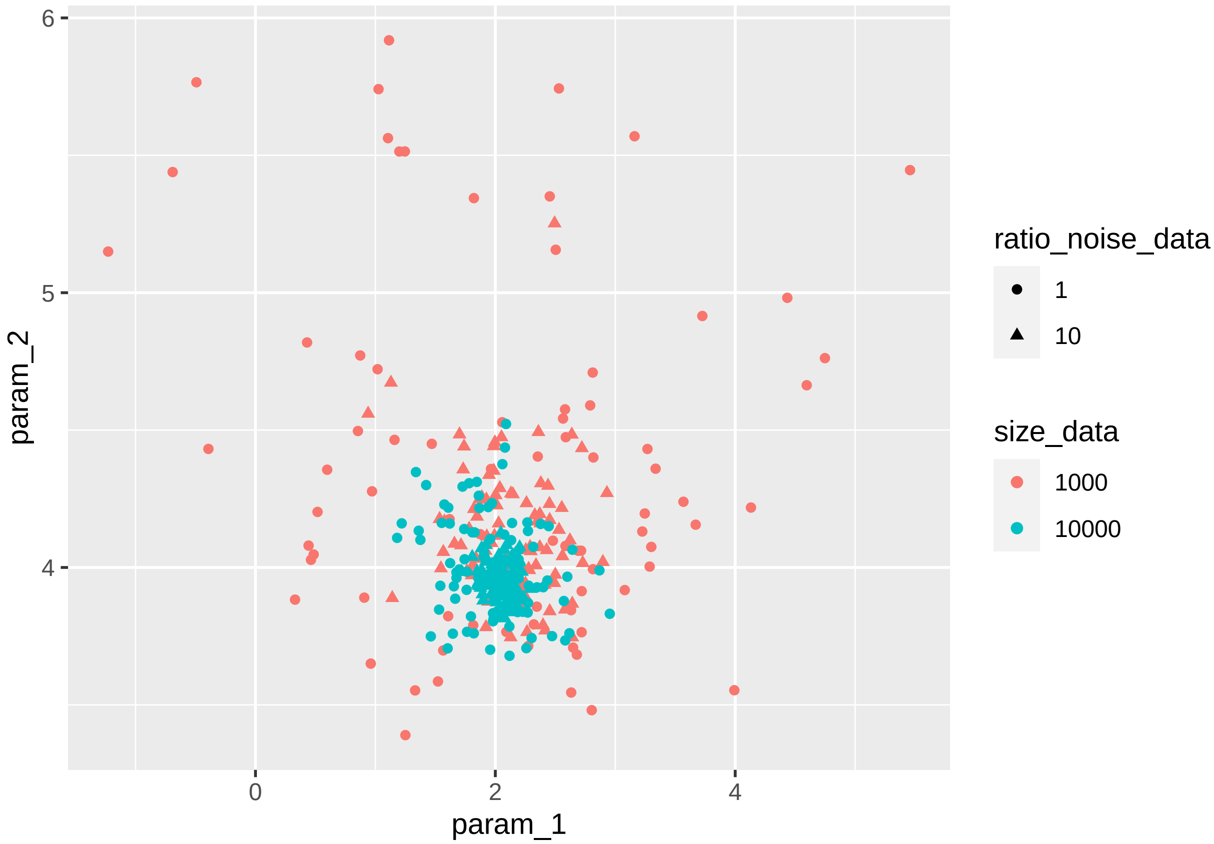
```
# METHODE MC MLE
mc_mle(x, n, c(mean(x),sd(x)), pm_barre)
```

```
## [1] 2.027152 3.885145
```

Crée un graphique qui prouve la convergence quand le ratio est de plus en plus élevé pour NCE.

```
df_mcmle_2 = data.frame(matrix(ncol = 4, nrow = 0))
colnames(df_mcmle_2) = c("param_1", "param_2", "size_data", "ratio_noise_data")

M = c(1000, 10000)


for (m in M){
  x = rnorm(m, 2, 4)
  psi = c(mean(x), sd(x))
  N = c(1, 10)
  for (n in N) {
    for (i in 1:100) {
       df_mcmle_2[nrow(df_mcmle_2) + 1, ] = c(mc_mle(x, m*n, psi, pm_barre), m, n)
    }
  }
}

df_mcmle_2$size_data = as.factor(df_mcmle_2$size_data)
df_mcmle_2$ratio_noise_data = as.factor(df_mcmle_2$ratio_noise_data)
```

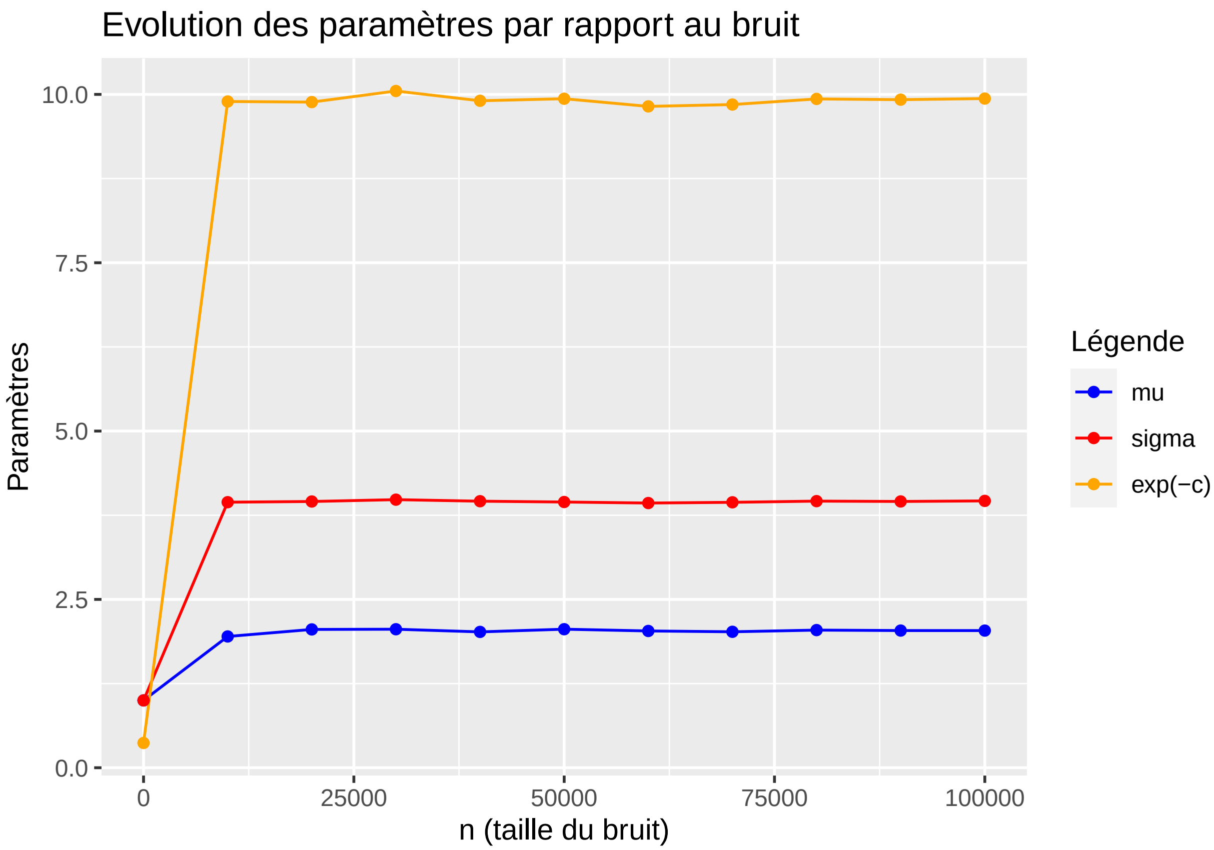
```r
df_mcmle_filt = filter(df_mcmle_2, param_2 <= 6 & param_2 >= 0 & param_1 >= -2 & param_1 <= 6)

ggplot(df_mcmle_filt, aes(x = param_1, y = param_2, color = size_data, shape = ratio_noise_data)) + geom
```




```r
# METHODE NCE
nce(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n)
```

```
## [1] 1.703905 3.610513 6.549622
```

```r
NCE_evol_params(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, 10, 10, c("mu", "sigma
```


Evolution des paramètres par rapport au bruit
10.0
7.5
5.0
2.5
0.0
Paramètres
0
25000
50000
75000
100000
n (taille du bruit)
Légende
mu
sigma
exp(−c)

3 Nouvelles approches

3.1 Bootstrap

```r
# Calcul de {size_boot} estimateurs par bootstrap
NCE_bootstrap = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_boot, labels) {
  m = length(x)
  theta_bootstrap = c()
  x_bootstrap = x
  for (i in 1:size_boot) {
    theta_bootstrap = append(theta_bootstrap, nce(x_bootstrap,
                                                  law_y,
                                                  params_y,
                                                  log_pm,
                                                  log_pn,
                                                  size_theta,
                                                  n))
    x_bootstrap = sample(x, size = m, replace=TRUE)
  }
  return(matrix(theta_bootstrap, nrow = size_theta))
}

# Plot la moyenne empirique des estimateurs bootstrap en fonction du nombre d'estimateurs
NCE_bootstrap_plot = function(matrix_theta_bootstrap) {

  # Formatage des données pour plot
  array_boot = 1:length(matrix_theta_bootstrap[1,])
  df = as.data.frame(cbind(t(rowCumsums(matrix_theta_bootstrap))/array_boot,array_boot))
  df_melted = melt(df, id.vars = "array_boot")

  # Plot
  plot_df = ggplot(df_melted, aes(x = array_boot, y = value)) +
  geom_line(aes(color = variable, group = variable)) +
  labs(title = "Evolution des paramètres par bootstrap",
       x = "Taille du bootstrap",
       y = "Paramètres",
       color = "Légende") +
  scale_color_manual(labels = labels, values = c("blue", "red", "orange"))
  print(plot_df)
}


# etude bootstrap de l'estimateur
bootstrap = function(matrix, alpha){
  return(data.frame(
    theta = matrix_theta_bootstrap[,1],
    biais = rowMeans(matrix_theta_bootstrap) - matrix_theta_bootstrap[,1],
    IC = rowQuantiles(matrix_theta_bootstrap, probs = c(alpha/2, 1-alpha/2))
  ))
}
```

```r
x_test = rnorm(1000,2,4)

matrix_theta_bootstrap = NCE_bootstrap(x_test, rcauchy, c(mean(x_test),sd(x_test)), log_pm, log_pn_cauc
```

```r
kable(bootstrap(matrix_theta_bootstrap, 0.05))
```

theta	biais	IC.2.5.	IC.97.5.
1.945617	0.0463546	1.672017	2.329352
4.086999	-0.0977920	3.780457	4.198468
10.023634	-0.0946162	9.237378	10.644296

3.2 Récursivité

Utilité : améliorer récursivement la précision de l'estimation via les estimations précédentes

```r
mc_mle_recursif_3 = function(x, n, psi, h, size_of_loop){

  m = length(x)
  Y = c(hasting(x, n, psi, h))
  H_x = c(h(x,psi))
  H_y = c(h(Y,psi))

  for (i in 1:size_of_loop) {

    L = function(theta){
      return(sum(log(h(x,theta)/(H_x/i))) - m*log(mean(h(Y,theta)/(H_y/i))))
    }

    psi = optim(
      par = rep(1,length(psi)),
      gr = "CG",
      control = list(fnscale=-1),
      fn = L
    )$par

    y = hasting(x, n, psi, h)
    Y = append(Y, y)
    H_x = append(H_x, h(x,psi))
    H_y = append(H_y, h(y,psi))

    print(psi)
  }

  return(psi)
}
```

```r
m = 10000
n = 10000
x = rnorm(m, 2, 4)
mc_mle_recursif_3(x, n, c(mean(x),sd(x)), pm_barre, 10)
```

```
## [1] 2.055095 4.045076
## [1] 1.764079 5.997641
## [1] 1.641461 7.942272
## [1]  1.59408 10.11521
## [1]  1.581055 12.442063
## [1]  1.799953 13.103222
```

```
## [1]  1.880269 14.363782
## [1]  1.908275 15.641773
## [1]  1.924142 16.323145
## [1]  1.920623 16.722022

## [1]  1.920623 16.722022
```

```r
nce_recursif = function(x, law_y, params_y, log_pm, log_pn, size_theta, n, size_of_loop){

  y = do.call(law_y, c(list(n),params_y))

  m = length(x)

  for (i in 1:size_of_loop){

    h = function(u, theta){
      return( 1 / (1 + n/m * exp(log_pn(u) - log_pm(u, theta))))
    }

    J = function(theta){
      return( sum(log(h(x, theta))) + sum(log(1 - h(y, theta))) )
    }

    theta = optim(
      par = rep(1, size_theta),
      gr = "CG",
      control = list(fnscale=-1),
      fn = J
    )$par

    print(theta)

    y = do.call(law_y, c(list(n),theta[-size_theta]))

  }

  return(c(theta[-size_theta], exp(-theta[size_theta])))
}
```

```r
nce_recursif(x, rcauchy, c(mean(x),sd(x)), log_pm, log_pn_cauchy, size_theta, n, 10)
```

```
## [1]  2.002001  4.001719 -2.306831
## [1]  2.085716  4.034019 -2.312118
## [1]  1.977974  3.983299 -2.309527
## [1]  2.115175  4.084126 -2.328714
## [1]  2.057584  4.023245 -2.302614
## [1]  2.031169  4.024756 -2.304978
## [1]  2.053019  4.007253 -2.305172
## [1]  2.034225  3.995113 -2.297533
## [1]  2.036713  4.049298 -2.323823
## [1]  2.095098  3.990496 -2.299116

## [1] 2.095098 3.990496 9.965366
```